Présentation du projet
Répertoire git public : https://github.com/Kamul-droid/projet_mlops.git

- **Titre** : MLOPSpour la Prédiction des Crises Cardiaques
- **Sous-titre** : Architecture, Implémentation et Contrôles de Qualité
- **Supinfo** : [Kamul Ali Nassoma Wattara- Sulaiman Fayyaz-Oumou Kanfana-Alex Alkhatib]
- **Date** : 14-11-24

Introduction au Projet

- **Objectif** :
    - Mettre en place un pipeline de données pour gérér le cycle de vie d'un projet IA.
- **Éléments Principaux** :
    - Chargement et prétraitement des données.
    - Exécution des contrôles de qualité.
    - Automatisation des étapes
    - Déploiement d'une API.

Framework

Le sujet se base sur un dataset HEART DISEASE.

Nature du problème : Classification

Qu'est ce qui est prédit :
La probabilité d'une personne soit diabétique ou non (0 pour Non et 1 pour Oui)

Type de données nécessaires :
Des données de patient atteinte de maladie cardiaque et ceux en bonne santé

Evaluation de la performance du modèle :

En classification de maladies, la **précision** (ou **accuracy**) n'est souvent pas suffisante pour évaluer la performance, car elle peut masquer des problèmes critiques, surtout si les classes sont déséquilibrées (par exemple, si la majorité des personnes n'ont pas de maladie dans le jeu de données). Voici les autres métriques importantes à considérer :

1. **Sensibilité (Recall ou Taux de vrais positifs)** :
   La sensibilité mesure la capacité du modèle à détecter correctement les personnes malades. Elle est particulièrement critique dans le contexte médical, car un faible rappel signifie que le modèle risque de ne pas diagnostiquer des individus réellement malades.

2. **Spécificité** :
   La spécificité mesure la capacité du modèle à identifier correctement les personnes non malades, ce qui est important pour minimiser les faux positifs. Une faible spécificité peut entraîner des sur-diagnostics, ce qui pourrait mener à des tests supplémentaires ou des traitements inutiles.

3. **F1-score** :
   Le F1-score est la moyenne harmonique de la précision (precision) et du rappel, et il est utile lorsque vous souhaitez équilibrer les deux aspects. Il est particulièrement précieux dans les cas où le coût des faux positifs et des faux négatifs est important.

4. **AUC - ROC (Area Under the Curve - Receiver Operating Characteristic)** :
   L'AUC représente la capacité du modèle à séparer les classes. Une AUC élevée indique que le modèle distingue bien les personnes malades des non-malades. La courbe ROC trace le rappel contre le taux de faux positifs, et l'AUC mesure l'aire sous cette courbe.

5. **Courbe de Précision-Rappel (Precision-Recall Curve)** :
   La courbe précision-rappel est souvent plus pertinente que la courbe ROC lorsque les classes sont déséquilibrées. Elle aide à analyser le compromis entre précision et rappel selon différents seuils de décision, ce qui est crucial pour ajuster le modèle selon des objectifs cliniques spécifiques.

Ligne de base :

Précision : >= 80 %

F1 : =>80 %

Architecture du projet :

- **Présentation de la Structure** :
  - *config/* : Contient les configurations du pipeline (fichier `config.yaml`).
  - *data/* : Contient les données source et traitées.
  - *scripts/* : Dossier principal contenant les scripts Python (e.g., `data_loader.py`, `preprocessing.py`, `quality_checks.py`, `server.py`).

- **Importance** :
    - Organisation modulaire pour faciliter la maintenance et l'extensibilité.

Script `data_loader.py`

- **Fonctions** :
    - `load_config()` : Charge le fichier de configuration.
    - `load_data(filepath)` : Charge les données brutes depuis un fichier CSV.
    - `save_data(df, filepath)` : Enregistre les données traitées dans un CSV.
    - **Utilité** : Automatisation de l'ingestion et du stockage des données.

Script `preprocessing.py`

- **Fonctions Principales** :
    - **Définition des Variables** :
        - Variables numériques, catégorielles, ordinales et binaires.
    - **Fonction `preprocess_data(df)`** :
        - Pipeline de prétraitement pour chaque type de variable.
        - Séparation des données en ensembles d'entraînement et de test.
- **But** : Préparer les données pour l'analyse et la modélisation.

Script `quality_checks.py`

- **Vérifications de Qualité** :
    - **Fonctions** :
        - `check_null_values(df)` : Vérifie les valeurs nulles.
        - `check_duplicates(df)` : Vérifie les doublons.
        - `validate_with_great_expectations(df, config)` : Valide les types et valeurs avec Great Expectations.
        - `delete_outliers(df, thresholds)` : Supprime les enregistrements qui dépassent les seuils prédéfinis.
- **Objectif** : Assurer la qualité et l'intégrité des données.

Contrôles de Qualité Avancés

- **Validations Spécifiques** :
    - *Age* : Vérification des valeurs entre 1 et 120.
    - *Sex* : Vérification des valeurs dans [0, 1].
    - *Cholestérol*, *FBS*, et *TRTBPS* : Vérification des valeurs entre des bornes définies pour chaque colonne.
- **Résultat** : Rapports de validation pour chaque test de qualité.

Détection et Suppression des Outliers

- **Fonction `delete_outliers`** :
    - Paramètre `thresholds` pour définir les limites.
    - Suppression des lignes où les valeurs dépassent les seuils.
- **Importance** : Nettoyer les données pour améliorer la qualité de l'analyse.

Déploiement avec FastAPI

- **API Endpoints** :
    - **`/ingest`** : Lance l'ingestion des données.
    - **`/quality-checks`** : Exécute les contrôles de qualité.
    - **`/data`** : Expose les 10 premières lignes du dataset traité.
    - **`/preprocess`** : Effectue le prétraitement des données.
- **Bénéfices** : Interface pour automatiser et exécuter les étapes du pipeline.

Voir le ReadMe pour la structure des dossiers, installations des dépendances …

Prefect : Orchestration des Pipelines (Les tâches Prefect sont regroupés dans des scripts commençant par « pipeline ».

1. **Définition des tâches** : Avec Prefect, chaque étape du pipeline de machine learning est encapsulée dans une fonction marquée par le décorateur **@task**. Ici, les principales tâches sont :
    - **ingest_data :** ingestion des données.
    - **run_quality_checks :** exécution des contrôles qualité.
    - **preprocess_data :** nettoyage et préparation des données.
    - **train_and_log_model :** entraînement et enregistrement du modèle.

Ces tâches sont indépendantes mais liées par des dépendances logiques.

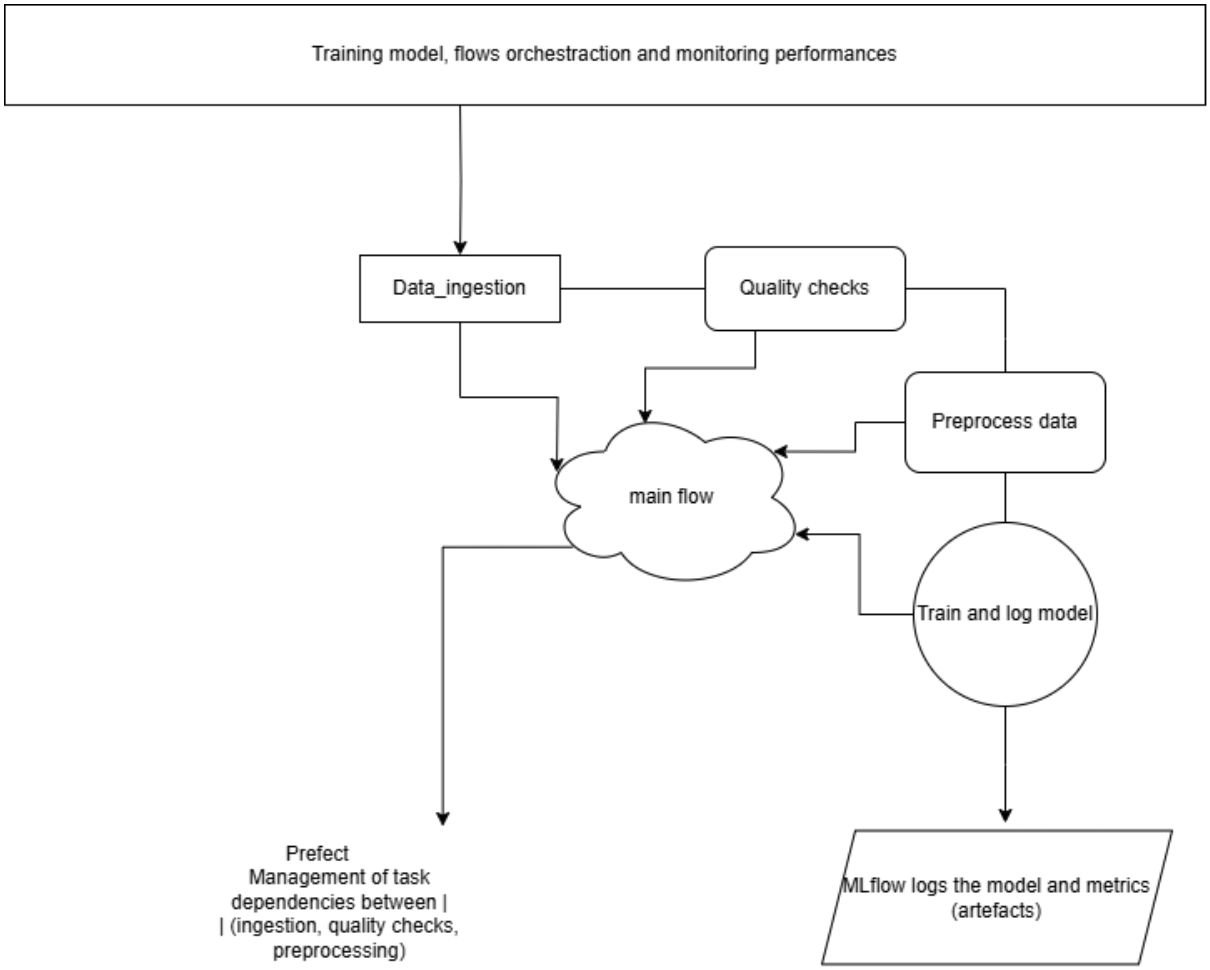
2. **Organisation et Enchaînement** : Les tâches sont exécutées dans un "flow" Prefect nommé main_flow, qui orchestre leur exécution séquentielle. Ce flow garantit que chaque étape se déroule dans le bon ordre. Par exemple, preprocess_data attend la fin de run_quality_checks avant de se lancer. Prefect gère ces dépendances et contrôle l'état d'avancement et les logs.
3. **Gestion des Logs et des Dépendances** : Grâce à Prefect, chaque tâche dispose de logs précis (via get_run_logger) qui sont facilement consultables. En cas d'erreur, Prefect facilite le debugging en isolant les logs et permet de relancer uniquement les tâches échouées sans refaire tout le pipeline.

MLflow : Suivi et Enregistrement des Modèles

1. **Tracking des Expériences** : À l'intérieur de la tâche train_and_log_model, MLflow est utilisé pour suivre l'entraînement du modèle. On commence par configurer MLflow pour capturer tous les hyperparamètres (comme n_estimators, max_depth) et les métriques de performance (précision, F1-score, log loss, etc.). Ces informations sont enregistrées dans un environnement MLflow centralisé, facilitant l'accès aux résultats et la comparaison d'expériences.
2. **Modèle et Model Registry** : Une fois le modèle Random Forest, logistic regression entraîné, il est enregistré dans MLflow. Le modèle, avec ses hyperparamètres et ses performances, est ajouté au Model Registry sous un nom unique. Cela permet de gérer facilement les versions de modèle dans MLflow, et le déploiement

Automatisation et Documentation : Grâce à l'intégration avec MLflow, chaque exécution du pipeline génère une nouvelle "expérience" dans MLflow, documentant automatiquement les configurations et performances du modèle.

Diagramme Orchestration des tâches et Entraînement de modèles



MLflow : Déploiement

MLflow propose divers moyens de déployer les modèles vers différents plateforme cloud ou en local pour l'inférence. Nous avons choisi de packager le modèle dans un container docker en raison de la facilité de sa maintenance et sa scalabilité verticale ou horizontale en combinaison avec Kubernetes.
Les scripts de déploiements sont ajoutés dans le dossier scripts du projet. Les fichiers DEPLOY_README.md et TEST_README.md comprennent les instructions pour le déploiement et l'inférence.